修 士 論 文

ネットワーク層プロトコルのハードウェア化による高速化手法に関する研究

指導教官　篠田 陽一 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

矢野大機

平成 11 年 2 月 15 日

目 次

図 目 次

表 目 次

第 1 章

はじめに

1.1 研究の背景

インターネットとは自律分散したネットワークの集合体である。ディジタルメディアのインフラストラクチャーとして発展してきた。このインターネットをもちいて、さまざまな通信がおこなわれている。インターネット上では 1 対 1 通信が基本となっている。

しかし、今後は利用形態の多様化によりあらたなトラフィックの発生が予想される。新しいメディアとして注目さているのは映像・音声などのマルチメディア通信である。

インターネットでは、ルータを介してネットワーク間を結んでいる。ルータが一方のネットワークから他方のネットワークへ情報を伝達している。インターネットでは、2 台のホストが通信をおこなうための方法を提供するために、その伝達処理に関する手順を分割し、レイヤ化をおこない各レイヤのプロトコルを詳細に定義している。このレイヤ化のモデルとして ISO の 7 層モデルを単純化した 4 層モデルが一般に用いられる。

図 1.1: 4 層モデル

各層は以下のように定義される。

層の名前	代表的なプロトコル	各層の意味
インターフェイス層	Ethernet	物理的な接続の確立
ネットワーク層	IP	相手ホストまでの到達性の確保
トランスポート層	TCP	ホスト間での伝達の制御
アプリケーション層	telnet , ftp	アプリケーションのふるまいを規定

このプロトコルを用いてパケットを転送することで通信をおこなう。ネットワーク間を結んでいるルータではネットワーク層プロトコルである IP が用いられてパケットの転送をおこなっている。

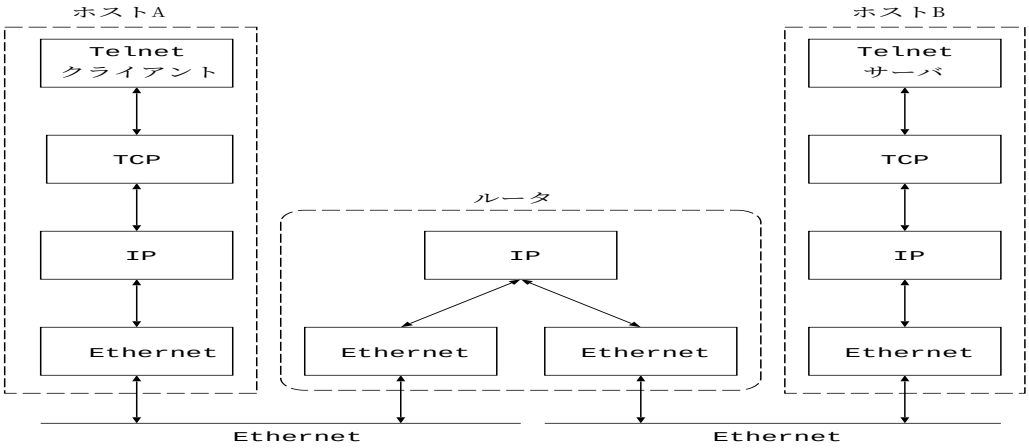


図 1.2: データ転送

基本的にはルータがホップバイホップにより、ベストエフォート型の転送をおこなっている。パケット転送をおこなうための基本ステップは以下のようになる。

- パケットから情報を読みとる
- ルーティングテーブルを検索する
- 送信すべきネットワークに送信する

ルーティングテーブルとは、ある宛先に対して次に送信すべきネットワークを示した表である。

しかし、映像や音声のデータを転送する場合にはパケットの優先順位付けなど帯域保証 (QoS) が必要となってくる。また、これらのデータを転送するためには、高速、広帯域な回線が必要になってくる。ネットワークの回線容量はここ数年で劇的に増大してきている。また QoS の実現も RSVP などのプロトコルが議論され、利用されてきてる。

しかし、増大する回線容量に対して、パケット転送に伴う処理の複雑化などにより、その回線を収容しているルータの伝送速度がボトルネックとなってきている。インターネットを速く、快適に利用するためにはスループットの向上をするための、なんらかの技術が求められている。スループットの向上のためには、いくつかの解決方法がある。

- 2 層でのルーティング
- ルータアーキテクチャの改善

ルータを用いないでパケットの転送をおこなう技術としては CSR などがある。ある閉じたネットワーク内でのデータ転送の技術としては有用である。しかし、インターネットは FDDI,X.25,ISDN,ATM など多様なネットワークを相互に接続し、それをひとつの巨大なネットワークに見せている。多様なネットワークを意識させずに相互接通信を可能にしているのがルータである。ルータの存在無しではインターネットは存在し得なくなる。そのためにはやはりルータ処理の高速化をする必要がある。

1.2 研究の目的

ルータにおけるプロトコル処理を高速化するためにはいくつかの方法が考えられる。

- ハードウェア化
- アルゴリズムの最適化
- プロトコルの改良

などがある。

ネットワーク層プロトコルは、信頼性のないコネクションレス型の転送をおこなう。データグラムが確実に到達したかどうかの保証がされないのである。また、??節でも述べたように、コネクションレス型のプロトコルである。つまり、データグラム間に順序は存在せず、各データグラムを独立に扱うことができる。つまり、2 つのデータグラム間での依存関係がない。順序や信頼性に関しては上位の層に依存する。これらより、一般的にネッ

トワークプロトコルは高度な並列性を有しているといえる。この並列性を活かすために、ハードウェアによるルータの実現が効果的であると考えた。

そこで本研究ではプロトコル処理のハードウェア化を考えていく。そのために必要な方法論の提案をおこなう。

1.3 本論文の構成

まず、本章である第 1 章で 研究の背景、目的を述べる。第 2 章でハードウェアを作成するための方法論を述べ、第 3 章で IPv6 に適用する。第 4 章ではハードウェアを作成するための知見を得るために、プロトコルスタックのハードウェアを設計し実装しその評価をする。第 5 章では提案した方法論の妥当性を検証する。第 6 章で結論として本研究をまとめる。

第 2 章

ハードウェア設計手法

本章では、まずプロトコル処理のハードウェア化をする際に用いる手法について検証する。次にその手法を用いたハードウェアを設計するための方法論を構築する。

2.1 ハードウェア化へのアプローチ

高度な並列性を有しているプロトコルは、並列処理可能なハードウェアで処理させるのが効果的である。本節では、既存のプロセッサで利用されているハードウェア処理の高速化技術を述べ、プロトコル処理のハードウェ化に有用な技術を検討する。

2.1.1 パイプライン

ハードウェアの性能向上の方法として、パイプライン処理と呼ばれる技術がある。パイプラインとは図 2.1 のように、複数の命令を少しずつずらし、同時並列的に実行する方法である。

命令実行のパイプライン化は、パイプラインステージとよばれる独立に動作可能なユニットで命令実行を各ステップ毎に分割する。パイプラインステージはクロックに同期したレジスタで分割される。各パイプラインステージにおける処理結果は、ステージ間に存在するレジスタを経由して、次のパイプラインステージに渡される。

パイプライン処理とは、ここの命令の処理時間を短縮するのではなく、全体のスループットを向上させる技術である。

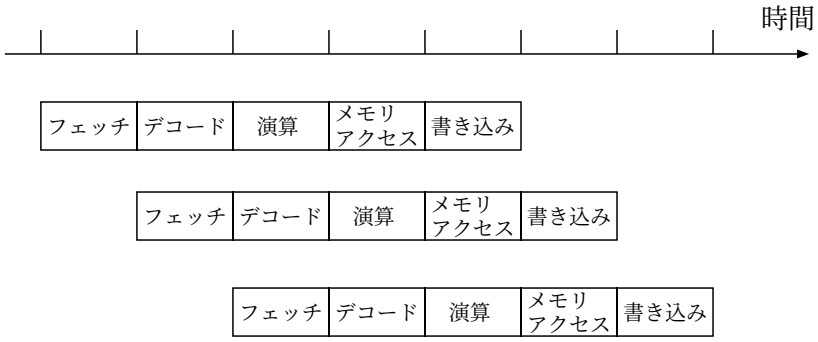


図 2.1: パイプライン

2.1.2 スーパースカラ

通常のパイプラインは、1 サイクルあたり 1 命令しか実行できない。これを、図 2.2 のように 1 サイクルあたり 複数命令実行できるようにしたのがスーパースカラパイプライン [2] である。

図 2.2: スーパースカラ

2.1.3 VLIW

スーパスカラ技術と似たような方式で超長形式機械命令 (VLIW[1])[2][4] という方式がある。この方式では図 2.3 のように、1 個の命令内に 1 個以上の並列実行可能な命令を組み込む。命令フォーマットにおける命令の位置がそれぞれ演算ユニットに対応している。VLIW 方式ではパイプライン、スーパーパイプライン方式に比べて命令数が減少する。

[1] very-long-instruction-word

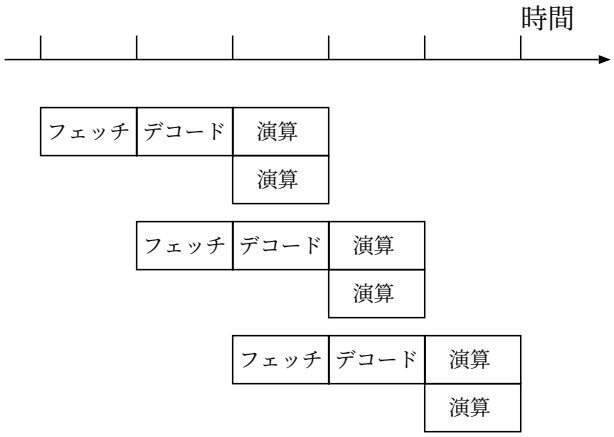


図 2.3: VLIW

2.2 ハードウェア設計上の制約

一般的にハードウェアを設計する場合に制約となる条件と、プロトコルをハードウェア化する際の制約となる条件について述べる。

2.2.1 ハードウェア設計上の制約

2.1 節で述べてきたようなパイプラインの設計には、以下に述べる 3 点が設計上の制約となる。これらの制約が回避できれば、性能向上が期待される。

データハザード: 実行する命令の大部分は何らかの結果を生成する。この値を後続命令が入力として使用する場合、データ依存関係があるという。データ依存関係がある命令が同時に実行された場合、他方の出力を必要とする命令は、入力が揃うまで実行を停止しなければならない。この状態をデータハザードという。

構造ハザード: 同一のハードウェア資源を 2 命令以上が同時に使用しようとした場合に発生するのが、構造ハザードである。

制御ハザード: ある分岐命令に後続する命令には制御依存関係があるという。この制御依存がある場合には、先行する分岐命令の結果によって後続命令が変更になる。つまり、分岐命令の結果がでるまでは、後続の命令が実行できないことになる。これを制御ハザードという。

2.2.2 プロトコルのハードウェア化における問題点

ネットワーク層プロトコルをハードウェア化する際に問題となる点を以下に述べる。

- パケットの大きさが巨大
  一つのデータとみなして扱うには、大きさが巨大すぎる。

2.3 ネットワーク層プロトコルの特徴

ネットワーク層プロトコルはコネクションレス型の転送をおこなうため、パケット間に順序が存在せず、データグラム間に依存関係がなく独立に扱える。この性質をもつため、プロトコルは高度な並列性を有しているといえる。またプロトコルはフィールドに分割でき、さらにこのフィールドは依存関係のないいくつかのグループに分けることができる。つまり、ネットワーク層プロトコルは、データハザードが発生しないことが保証され、かつ同時に複数の実行が可能な 1 つの巨大な命令とみなせる。

2.1.3 節で VLIW 方式の欠点として命令の並列度が高くない場合には、効率的にパイプラインを利用できないことを述べた。しかし、ネットワーク層プロトコルには各フィールドを依存関係がないグループに分割できる。これらのグループは同時実行可能であるため、1 つのグループを 1 つの命令とみなせば、命令の並列度を高く保つことができる。

つまり、ネットワーク層プロトコルは、VLIW 方式を用てパイプライン化されたハードウェアによる処理が可能である。

2.4 ハードウェア化手法

一般にハードウェアは 図 2.4 の手順で設計される。

機能設計

ハードウェアの仕様を決定する。本研究では、この機能を設計するための方法論を提案する。

論理/回路設計

仕様に従い動作記述をおこなう。シミュレータを用いることで、記述された回路の動作確認をおこなう。

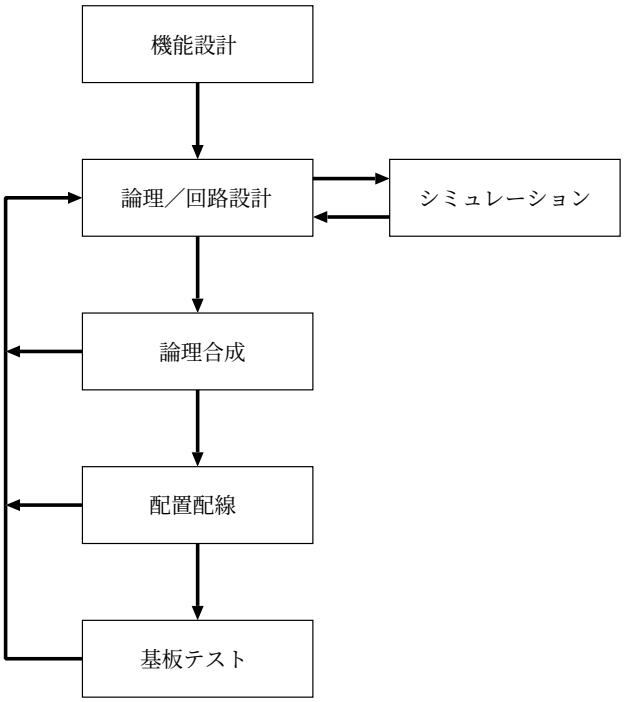


図 2.4: 設計フロー

論理合成

ハードウェア記述言語で書かれた記述より、論理回路を生成する。

配置配線

対象とするチップにあうように、ゲートの配置、配線を決定する。回路数の削減などが必要ならば、論理/回路設計段階にもどり、再設計をする。

基板テスト

実際の基板を作成し、テストをする。

2.5 提案手法概要

ネットワーク層プロトコルを VLIW 的に扱うには、プロトコルの各フィールドを依存関係のないグループに分割する必要がある。そのために、各フィールド毎にモジュールと

呼ぶ単位に処理を分割し、そのモジュール間での依存関係を抽出する。依存関係があるモジュール群を一つのグループとし、そのグループ毎にパイプライン化をする。最後に、各々パイプライン化されたグループの最適化をし、ひとつのハードウェアとして完成させる。

ネットワーク層プロトコル処理には大きく分けて2種類ある。正常にパケットを転送できる場合とICMP処理を発生する異常処理の場合である。さらに、ネットワーク層プロトコルも基本ヘッダとオプションヘッダの2種類に分けることができる。正確にいえばオプションヘッダは複数あるが、基本ヘッダ以外をすべてオプションヘッダとする。ハードウェア化する際には、これら4種類の組合せをそれぞれ独立に考えていく。それぞれの場合でパイプラインを設計し、それぞれのパイプラインの最適化をおこなう。

本研究では以下の手順でハードウェアの設計をする。

- ハードウェアに必要な情報の抽出
- 処理単位の分割
- データ依存関係の把握
- イベント依存関係の把握
- モジュールの処理時間の評価
- パイプライン化
- パイプラインの最適化

2.6 提案手法

2.5節で述べた各手順をより詳細に定義する。

2.6.1 ハードウェア化に必要な情報の抽出

ネットワーク上のコンピュータは厳密に定義されたプロトコルを用いて通信をおこなっている。プロトコル仕様はISOC[2]のIETF[3]およびIAB[4]が中心となり技術検討をおこな

[2]Internet Society

[3]Internet Engineering Task Force

[4]Internet Architecture Board

い、ドラフト段階のものも含めRFC[5] とよばれるドキュメントとして文章化されている。

ハードウェア化するプロトコルの仕様はRFCに定義されている。ハードウェアを設計するために次の仕様が必要である。

- 各フィールドのビット幅
- 各フィールドの位置
- 各フィールドに代入される値

2.6.2 フィールド処理の分割

入力パケットのフィールド値を入力値とし、なんらかの処理をし、出力パケット生成する。この過程で、各フィールドがおこなう動作を決定する。各フィールドに対しいくつかの動作をする必要があり、各々の動作について、その動作に必要な入力値を決定する。入力値とは、他のフィールド値や、外部のデータベースである。さらに、これらの動作を以下に述べる基準に従い分割する。この分割した動作を1つの動作単位とする (以下モジュールと呼ぶ)。

- 2つ以上の異なる値を出力しない
- 処理途中で他フィールドの値を参照しない
- 2つ以上のデータベースを参照しない

値を出力しない動作については、入力された値がそのまま出力されるとして考える。

図 2.5: 処理単位分割

図2.5では何らかの処理を円として表している。この図で説明すると、破線で囲まれた部分を一つの処理としてとらえるのではなく、円で表される処理を一つの処理単位として考える。

[5] Request For Comment

2.6.3　データ依存関係の把握

モジュール間の依存関係を把握するために、データの流れに注目したフロー図を作成する。

データフロー図の概略を説明する。モジュールは楕円として描かれ、動作名を楕円内に示す。各モジュールは入力、出力の矢印を持つ。各矢印は矢印の向きに値を運ぶことを示している。モジュールに入力された値と出力された値が異なる場合には、矢印上に出力された値に対するラベルをつけることもある。さらに、モジュールの位置により、そのモジュールの動作が開始されるタイミングを示す。

以下にデータフロー図の作成手順を示す。

1. 全モジュールを同じタイミング上にならべる
2. あるモジュール (a) の出力を入力として必要とするモジュール (b) を探す
3. モジュール (a) とモジュール (b) 間を矢印で結ぶ
4. そのモジュール (b) をモジュール (a) の動作タイミングより後ろにづらす。これと同時に、モジュール (b) と線で結ばれており、かつモジュール (b) より 後の動作タイミングであるモジュールすべてを、矢印で結ばれた各モジュール間の関係を保ったまま動作タイミングが後になるようにづらす。
5. 3-5 を 全モジュールに対しておこなう

図 2.6 に例を示す。モジュール B とモジュール C の間にデータの依存関係があった場合である。

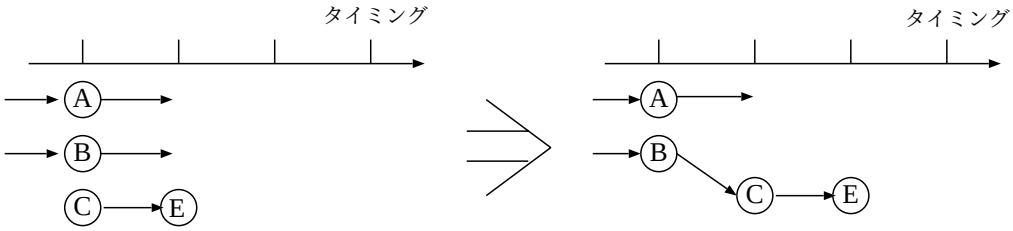


図 2.6: データフロー図

以下、このステップで作成した図に対して変更を加えていくことで、パイプラインステージの設計をおこなっていく。

2.6.4 イベント依存関係の把握

動作が他のモジュールの計算結果に依存するモジュールの検出をおこなう。ここで他のモジュールの計算結果に依存するとは、処理の分岐がおきることである。あるモジュール (a) の結果によりあるモジュール (b) の動作が異なってくる場合に、イベント依存関係があると定義する。

この関係の把握をするために、以下のことをする。

1. 分岐を起こす条件は、比較演算によって決定される。そのため、比較演算操作をおこなうモジュールを見つける。

2. それらの各モジュールに対し、入力として考えられる値を代入し、代入した値により影響を与える他のモジュールを探す。

ここでは、通常処理とエラー処理を分けて考えていく。
イベント依存があるモジュールは更に以下の 2 つにわけられる。

1. 他のモジュールの結果を参照しなくては処理の実行ができない。(投機的実行が不可能)

2. 他のモジュールの結果を参照しないで処理の実行ができる。この場合は、他のモジュールの結果によっては処理自体が無駄になる場合がある。(投機的実行が可能)

2 に関しては並列動作が可能であるが、1 に関しては並列動作ができない。そこで、1 の関係にあるモジュールに対して 2.6.3 節の 4,5,6 の操作をおこなう。

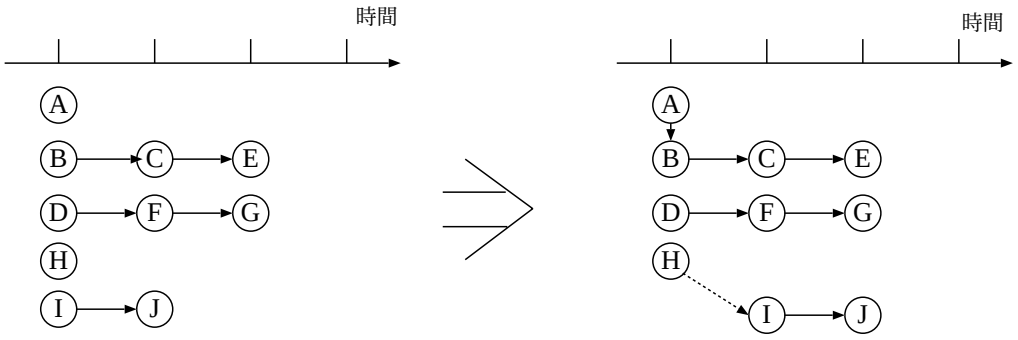


図 2.7: イベント依存

A と B,H と I に依存関係があるが、B は投機的実行が可能であり、I は投機的実行が不可である場合。

イベント依存関係があるモジュールを破線で結ぶこととする。

2.6.5 モジュールの処理時間の評価

演算時間は、実装する方法、製造技術などにより大きくかわってくる。

各モジュールの演算時間の計算には、以下に述べる要素を考慮する。また、コスト関数を別途定義し評価をおこなう必要がある。

- 製造技術
- データのビット幅
- 動作クロック
- メモリ速度
- I/O の速度
- クロック・スキュー

これらの条件を評価し、各モジュールの演算時間を計算する。

この値を基に、第 2.6.3 節で述べた図に対し各モジュール間を引き延ばす。

この評価関数で評価された値をもとに比較の基準となる演算を決定する。各モジュールがその基準演算に対してどのくらい演算時間がかかるのかを検討する。もしくは、各モジュールが処理終了までに何クロック必要とするかの検討をする。この処理終了までの時間とは、値の入力開始からなんらかの値の出力が終了するまでの時間である。

また、1 回の処理時間が毎回変化するようなモジュールの場合は、何らかの方法で処理時間を決定する必要がある。例えば処理時間の平均値といったものを用いる必要がある。この時間内に処理が終了しなかった場合は例外処理となる。

2.6.6 パイプライン化

データフロー図に対し、イベント依存関係、演算時間の評価を加えた図で、線で結ばれているモジュール群を 1 つのグループとして扱う。そのグループがパイプラインの一つのラインを構成する。図 2.8 では、グループ化されたものを破線で囲んでいる。

これらのグループに対して、入力タイミングの考察をする必要がある。ヘッダデータの入力に対して、2.6.5 節で述べた方法により値が入力される時間を計算する。

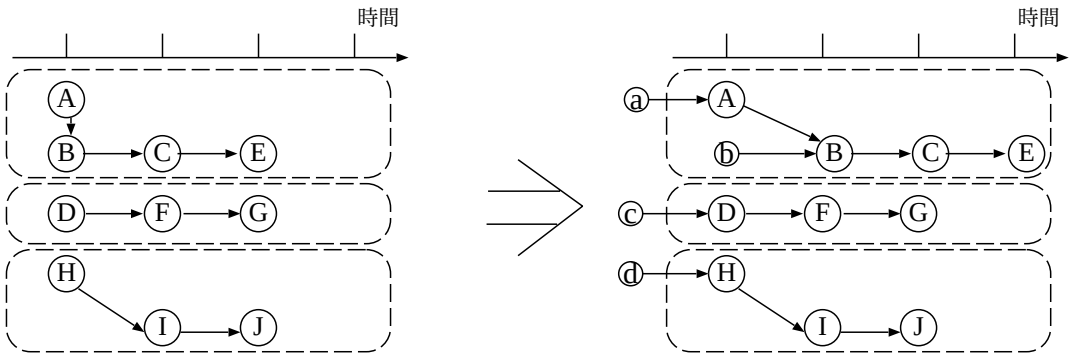


図 2.8: パイプライン化

2.6.7 パイプラインの最適化

各グループとして構成されたパイプラインを、ひとつのパイプラインとして合成する。基本パスを基準に最適化する。

次の条件をみたすように合成する。

1. 各モジュールの動作時間を最も遅いモジュールと同じ処理時間にする
2. 同じモジュールは同じ時間で利用されるようにする図 2.9(I)
3. 出力がおなじタイミングとなるようにする
4. 基本パス (図 2.9 では破線で囲まれたパス) 内のモジュールよりも、モジュールの駆動タイミングを早くしなければならない場合は、以下の基準に従う。

   - 動作タイミングを早くしなければならないモジュールが頻繁に駆動されない場合

     基本パスに対して同じタイミングで動作させる。そのモジュール以降のモジュールのスタートを基本パスにあわせる。つまり、稀に動作するモジュールがパスに含まれる場合、そのラインが完了するまでに 2 パケット分の処理時間がかかることになる。図 2.9(II)

   - それ以外の場合

     基本パスの実行をおくらせる。図 2.9(III)

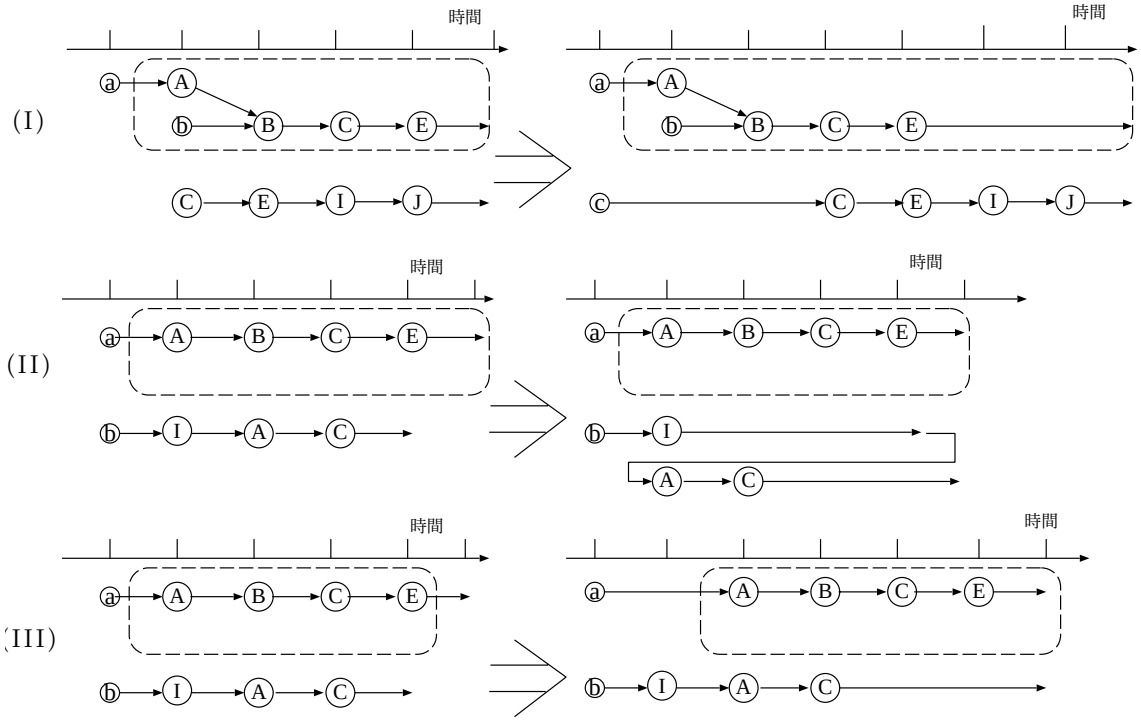

(I)
時間
a
A
b
B
C
E
C
E
I
J
時間
a
A
b
B
C
E
c
C
E
I
J
(II)
時間
a
A
B
C
E
b
I
A
C
時間
a
A
B
C
E
b
I
A
C
(III)
時間
a
A
B
C
E
b
I
A
C
時間
a
A
B
C
E
b
I
A
C

図 2.9: 最適化

第 3 章

IPv6へ適用

提案した手法を IPv6 に適用する。まずIPv6 の概要を述べ、次に実際に提案手法を適用する。

3.1 IPv6の概要

この節では IPv6 の特徴を簡単に説明する。

アドレス空間の拡張

現在主に用いられているネットワーク層プロトコルである IPv4 は、32 ビットのアドレス空間を持つ。しかし、IPv6 では これを 128 ビットに拡張し、アドレス空間を 2^{96} 倍に拡大した。

ヘッダフォーマットの簡略化

IPv4 を運用してきた経験を基に、使われないヘッダを省き、実時間通信などで有用なフローラベルを導入した。

図 3.1 と図 3.2 を比較すると分かるように、IPv6 では アドレス長が 4 倍になったにもかかわらず、全体として大きさは 2 倍になっただけである。

主に次の簡略化がおこなわれた。

- ルータにおける分割処理の禁止
- チェックサムの除去

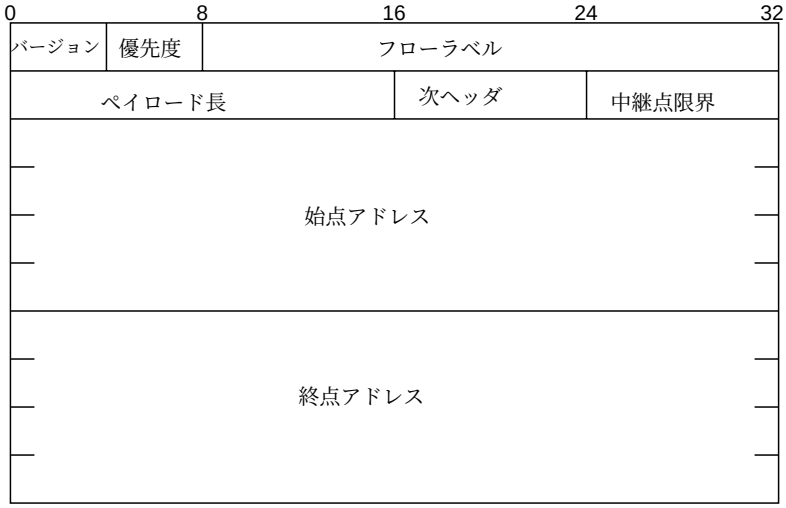


図 3.1: IPv6 基本ヘッダ

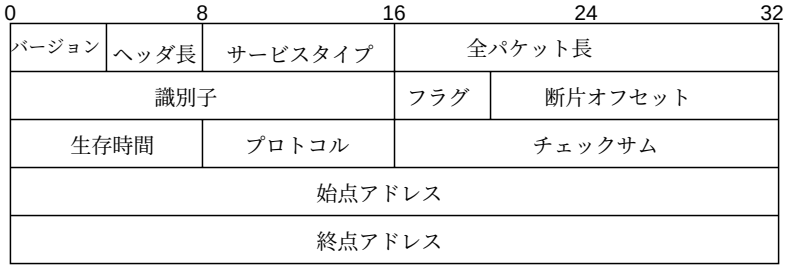


図 3.2: IPv4 ヘッダ

数珠つなぎヘッダ

IPv4 では IP ヘッダの直後に TCP 等のペイロードが続く。しかし、IPv6 では IP ヘッダの直後に任意の数の拡張ヘッダを挿入できる。拡張ヘッダとしては以下のものが定義されている。

- 中継点オプションヘッダ
- 経路制御ヘッダ
- 断片ヘッダ
- 認証ヘッダ

- 暗号ペイロード
- 終点オプションヘッダ

このように、ヘッダ毎に機能を分けることにより、IPv6 の基本ヘッダを固定長にすることができる。また、任意のヘッダを挿入できるために、拡張性に優れているといえる。

セキュリティー機能の強化

現在のプロトコルではセキュリティーの機能はあまり考慮されていない。しかし、ネットワークの普及における不正アクセスの増加などにより、セキュリティーの重要性が高まってきた。そのため、IPv6 では標準でセキュリティーの機能を提供している。

3.2 IPv6プロトコル処理ハードウェアの実現

IPv6 プロトコルスタックに提案手法を適用する。

3.2.1 ハードウェア化に必要な情報の抽出

IPv6 は RFC2460[8] に仕様が、RFC2373[9] にアドレスアーキテクチャの仕様がかかれている。

IPv6 の構造は 前節でも述べたように、図 3.1 のような構造をしていることがわかる。

各フィールドの意味

- バージョン

  インターネットプロトコルのバージョン番号

- 優先度

  データグラムの優先順位

- フローラベル

  あるフローに属するパケット群を識別する

- ペイロード長

  ヘッダの後に続くデータ長

- 次ヘッダ

  基本ヘッダの次に現れるヘッダ番号、この番号はRFC1700[6] で定義される。

- 中継点限界

  パケットが消滅するまでの中継点数

- 始点・終点アドレス

  始点、終点となるホストのインターネットアドレス

IPv6 のアドレス体系には次の 3 種類が存在する。

- ユニキャストアドレス
- マルチキャストアドレス
- エニーキャストアドレス

ユニキャストアドレスはさらに細かくわかれており、また、これらのアドレスに対しては、ある値が初期値として割り当てられている。それらの詳細については 付録 A で述べる。

3.2.2 処理単位の分割

3.2.1 節で述べた各フィールドを解釈し処理するために、おこなわなければならない処理について考察する。おこなわなければならない処理には大きく分けて 2 種類が存在する。

- 正常動作する時
- エラー処理をおこなわなければならない時

である。本手法ではこれら 2 つの処理を別々にとらえハードウェアを設計する。以下、正常動作する場合について考察していく。

トラフィッククラス、フローラベルに関しては仕様が確定していないため、本論文でハードウェア化の対象としない。

また、ここで書き出した処理はIPv6 ヘッダが、基本ヘッダのみであった場合におこなう処理である。

- Version(4 ビット)

  ○ 6 であるか

- ペイロード長 (16 ビット)
  - ○ リンク MTU よりも小さいか
- 次ヘッダ (8 ビット)
  - ○ TCP や UDP などの 上位レイヤであるか
- 中継点限界 (8 ビット)
  - ○ 1 以上であるか
  - ○ 1 を引く
- 始点アドレス (128 ビット)
  - ○ マルチキャストアドレスか
  - ○ リンクローカルアドレスか
- 終点アドレス (128 ビット)
  - ○ マルチキャストアドレスか
  - ○ リンクローカルアドレスか
  - ○ 自分宛か
  - ○ 経路検索

以上のことをおこなう。

また、第 2.6.2 節で述べた条件を上記の処理に当てはめてみると、経路検索処理が 2 つのステップに分解できる。

- インターフェイスインデックス (IF インデックス) をひく
- インターフェイスアドレス (IF アドレス) をひく

3.2.3 データ依存関係の把握

各処理について詳細に見ていくと、

- リンク MTU との比較
- IF アドレスの検索

の 2 つの処理が、IF インデックス検索処理の結果を必要とする。

つまり、これら 2 つのモジュールは IF インデックス検索処理よりも、時間的に遅いタイミングで実行されなければならない。

これらの条件を加味してデータフロー図を作成すると、図 3.3 のようになる。

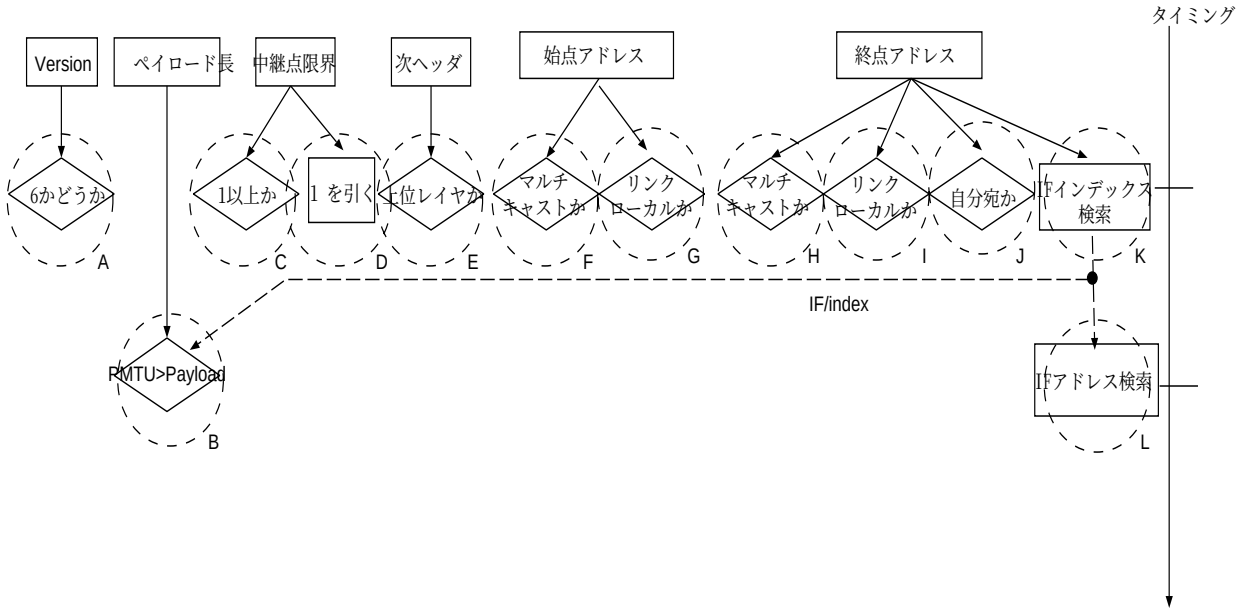


図 3.3: フロー図

破線の円で囲んであるものが 1 つのモジュールである。また、以後各モジュールを破線の円の右下に書いてある記号で略記する。

3.2.4 イベント依存関係の把握

比較演算をおこなうモジュールは

a,b,d,e,f,g,h,i,l

である。この中で他のモジュールに結果が参照されるのは

b,g,h

である。

これらを参照するモジュールは

c,i

である。これら c,i は 2 つに分けられる。表にまとめると 以下のようになる。

表 3.1: イベント依存関係

参照もと	参照さき	投機的実行可能か
c	d	投機的実行可能
h　i	j	投機的実行不可能

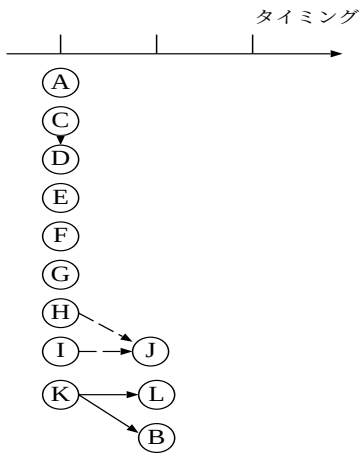


図 3.4: イベント依存

3.2.5　モジュールの処理時間の評価

本来であれば、2.6.5 で述べた条件を用い、各モジュールの処理時間を評価しなければならないが、本論文では対象となるハード ウェアの実現条件を設定していないため、次のように仮定してモジュールの処理時間を評価する。

- データの転送等の遅延は無視できる程小さい
- 四則演算、比較演算時間 1
- データ検索、メモリ参照時間 5
- データベース検索は 1 度のメモリアクセスで終了する

以上の条件で各モジュールを評価すると表 3.2 のようになる。

表 3.2: 各モジュールの演算時間

モジュール名	実行処理 実行回数	処理時間合計
a	比較演算 1	実行時間 1
b	比較演算 1	実行時間 1
c	比較演算 1	実行時間 1
d	減算 1	実行時間 1
e	比較演算 3	実行時間 3
f	比較演算 1	実行時間 1
g	比較演算 1	実行時間 1
h	比較演算 1	実行時間 1
i	比較演算 1	実行時間 1
j	比較演算 1 、データベース検索 5	実行時間 6
k	データベース検索 1	実行時間 5
l	データベース検索 1	実行時間 5

3.2.6 パイプライン化

演算時間の評価を図に加える。この図で線で結ばれているモジュール群がひとつのパイプラインステージを構成する。ここでは入力として必要な値は全て同時にアクセスできるものとする。図 3.5 にその結果を示す。

3.2.7 パイプラインの最適化

同様な手順で、IPv6 基本ヘッダ＋経路制御ヘッダの場合の処理について考えると、図 3.6 のようなパイプラインが構成される。

追加されたモジュールのみ説明する。

図 3.5 と図 3.6 であらわされた 2 つのパイプラインの最適化をおこなうと図 3.7 となる。

この 2 つのパイプラインの最適化で第 2.6.7 節で述べた 4 の条件に当てはまるのがモジュール S である。この最適化では、経路制御ヘッダ処理が頻繁におこなわれると仮定して最適化をおこなった。

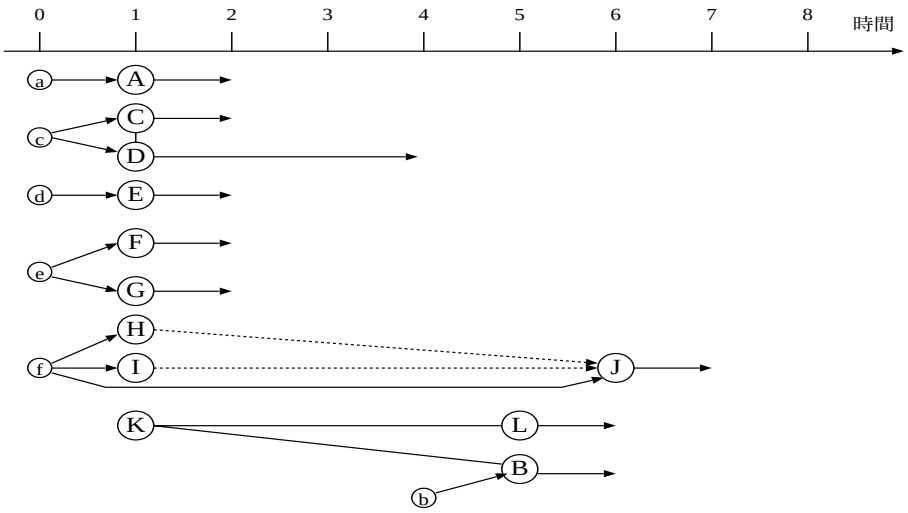


図 3.5: パイプライン化

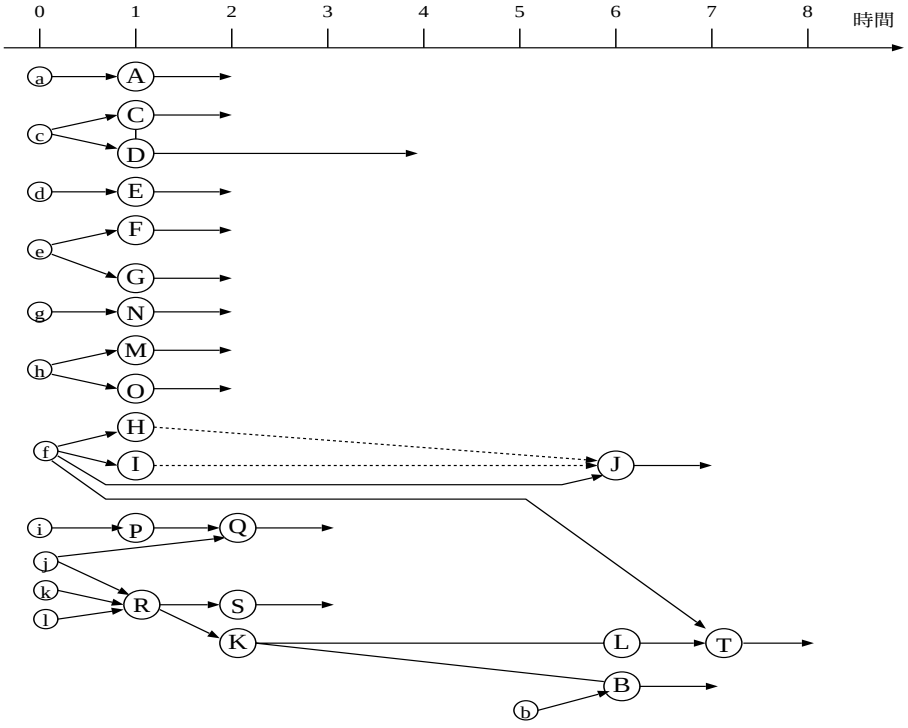


図 3.6: 基本ヘッダ＋経路制御ヘッダ

表 3.3: あらたに加えられたモジュール

モジュール名	動作	入力値 (用いた略記)
M	上位レイヤか	次ヘッダ (g)
N	偶数か	拡張ヘッダ長 (h)
O	2 で割る	拡張ヘッダ長 (h)
P	0 かどうか	経路制御型 (i)
Q	ヘッダ長より大きいか	中継点残数 (j)
R	アドレスの選択	アドレス (k)(l)
S	マルチキャストか	
T	アドレスの書き出し	

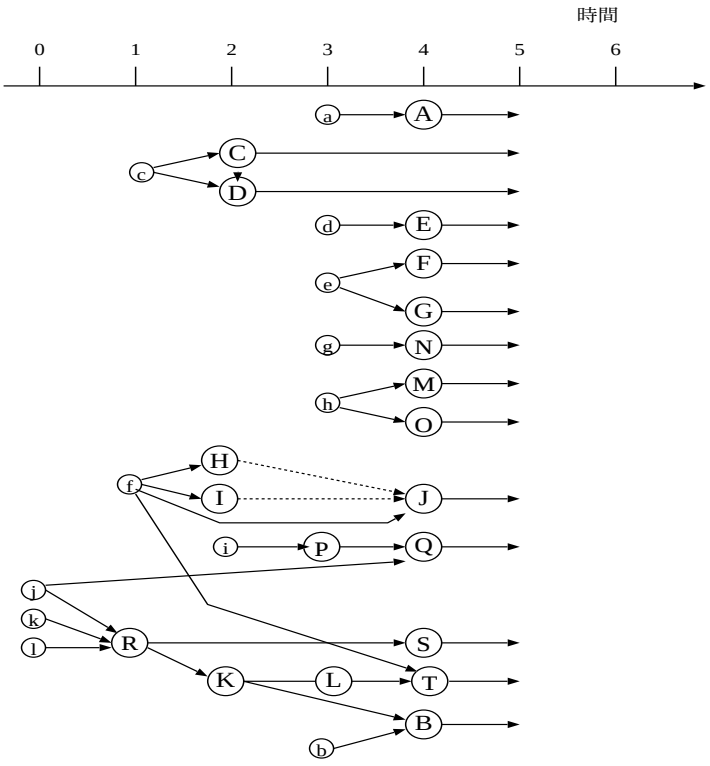


図 3.7: パイプライン最適化

第 4 章

プロトコル処理モジュールの設計と実装

プロトコル処理をハードウェア化の方法論を構築するためにの知見を得ることを目的に、IPv6 のプロトコル処理部の一部を FPGA を用いた実装をおこなった。その実装の概要を示す。

4.1 実験環境

以下のツールを用いて実装をおこなった。

FPGA ボード

Xilinx 社の FPGA ボード WILD-FORCE[1]

図 4.1 に WILDFORCE のブロック図を示す。

CPE1、CPE2、CPE3、CPE4 が実際に書き込み可能な FPGA である。そこに ロジックを書き込み動作させる。データは PCI バスから FIFO0、FIFO1、FIFO4 へ入力し、各 CPE に渡される。

シミュレーション ソフトウェア

シミュレーションは Model Techmology 社の ModelSim PE/PLUS[2] を用いた。

ModelSim VHDL は、IEEE 1076-1987 and 1076-1993 VHDL standards をサポートしている。

[1] http://www.annapmicro.com/wfhtm.html#PCI

[2] http://www.model.com/

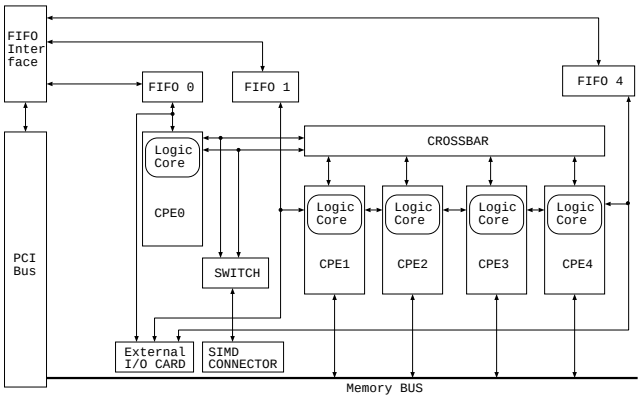


図 4.1: WILDFORCE ブロック図

合成・配置配線 ソフトウェア

Synplicity 社の Synplify[3] を用いた。

4.2 実装目的

プロトコルのハードウェア化の方法論を確立するにあたり、ハードウェア設計手法の知識を得るために、IPv6 プロトコルスタックの一部のハードウェア化を考察し、4.1 節で述べた環境下で実装し、ハードウェアに対する知見を得た。そして、ここで得られた知見をもとにハードウェア化の方法論を構築した。

4.3 ハードウェア仕様

4.3.1 入出力

実験に用いたボードは PCI バスを通して、32 ビット幅の FIFO に対して 入力をおこなう。PE1 が FIFO1 より値を読みだす。

出力に関しては PE4 が FIFO4 に対して書き出し、FIFO4 の値を PCI バスをとおして読みだす (図 4.1 参照)。

[3] http://www.synplicity.com/

4.3.2 動作仕様

IPv6 のヘッダ構造は、図 A.2 に示したように数珠つなぎ構造をとっている。このようなヘッダ構造としたことで、オプションヘッダの挿入を容易にした。しかし、必要なヘッダ位置を知るためには、IPv6 基本ヘッダより 線形に検索しなければならなくなった。あるヘッダから必要な情報を得るためには、必ずこの検索動作をおこなわなければならない。

この動作を簡略化するために、本ハードウェアでは 図 4.2 に示すテーブルを出力する。また、検索動作を、このテーブルを参照する動作におきかえることで、処理速度の向上を計った。

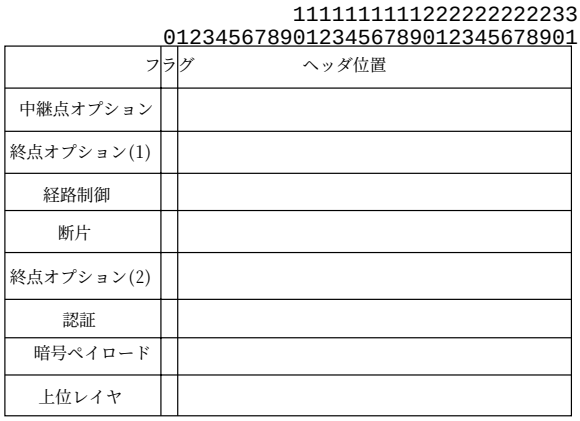


図 4.2: 出力テーブル

フラグは 1 ビットで、そのヘッダが、存在するかしないかをあらわす。

ヘッダ位置は 31 ビットで、そのヘッダの IPv6 基本ヘッダからの距離をあらわす。

4.3.3 処理アルゴリズム

FIFO が 32 ビット幅であるので、32 ビットずつヘッダを読み込む。テーブルを作成するために必要となる部分は、各ヘッダの 次ヘッダ の部分と 拡張ヘッダ長の部分である。

これらの情報をボードにあるメモリ上で作成し、テーブルが完成したら、FIFO へ書き出す。

1. IPv6 基本ヘッダの次ヘッダフィールドから次ヘッダを読みとる
2. 次ヘッダの先頭まで読み込む

3. 次ヘッダフィールドと、拡張ヘッダ長フィールドを読みとる
4. 拡張ヘッダ長から計算される、次ヘッダの先頭まで読みとる。と同時に、テーブルに書き出す
5. 次ヘッダが 上位レイヤになるまで上記 3,4,5 をくりかえす
6. メモリ上に作成されたデータを FIFO へ書き出す

以上の処理をおこなう。

4.3.4 ステージ分解

前節で述べた処理をステージに分解することで、パイプライン的に処理できるようにする。

読み込みステージ

クロック毎に FIFO から 32 ビットずつデータを読み込む

解析ステージ

入力されたデータを解析。必要なデータを次ステージにわたす。

整形ステージ

次ヘッダ番号と、拡張ヘッダ長より IPv6 基本ヘッダからの位置を計算し、メモリ上にテーブルを作成する。

出力ステージ

全ヘッダの解析が終了したら、メモリ上に展開されているテーブルを FIFO に出力。メモリの初期化をおこなう。

4.4 評価

実装したハードウェアに対して、実トラフィック上から採取したデータを入力し、テーブルが出力されるまでの時間を測定した。

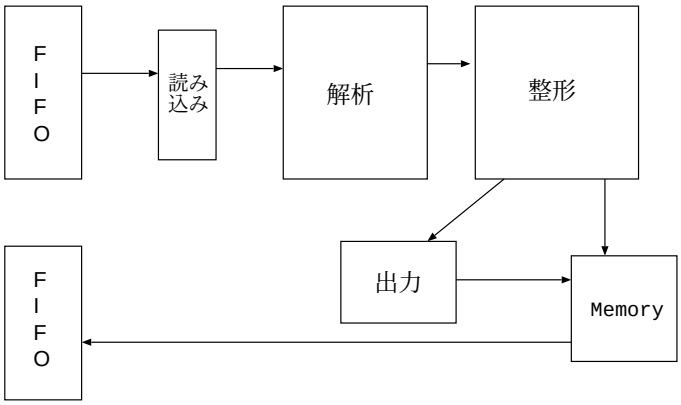


図 4.3: ブロック図

4.4.1 処理速度

シミュレーションをおこない、動作速度を測定した。WILDFORCE は クロック 20MHz で動作させることができる。1 クロックは約 48ns である。IPv6 基本ヘッダのみの場合は、FIFO からデータを読み始めてから、テーブルが完全に FIFO に出力されるまで、1 マイクロ秒を要した。

次に 128 ビット長の拡張ヘッダがついた場合には、約 2 マイクロ秒を要した。

ハードウェアの処理速度との比較をおこなうために、同処理をソフトウェアでおこなった場合の速度を検証した。

対象とした 処理は BSD/OS 3.0 をベースに開発された IPv6 プロトコルスタックである。このプロトコルスタックが組み込まれている カーネルでは、IPv6 のパケット転送速度は 約 5000 pps である。一つのパケット処理時間は 200ms となる。このうち ヘッダを解析する処理部のステップ数は約 400 ステップ存在する。このカーネルを 200 MHz の周波数で動作させると、400 ステップの処理に約 3 マイクロ秒要する。

ハードウェア化をおこなうことで、約 30%の高速化が可能になった。

4.5 妥当性の考察

本研究で実装したハードウェアは IPv6 プロトコル処理の一部を実装したに過ぎない。パケット処理全体としては、あまり速度向上が望めなかった。これは、今回実装した部分がヘッダ処理部の一部でり、この部分の処理時間が 0 であるとしても、全体の処理時間の向上は望めないことが原因である。しかし、ヘッダ処理部だけをみれば 30%高速化されて

いる。さらに、他の部分がソフトウェアで処理されており、ハードウェアの処理と比べて低速である。この影響を受けてヘッダ処理部の処理速度の向上は30%にとどまったといえる。これらより、プロトコル処理全体をハードウェア化することで、効率的な高速化が実現できると予想される。

第 5 章

考察

プロトコルのハードウェア化に対する議論をする。さらに、ハードウェアに適したプロトコルについて議論をする。

5.1 プロトコルのハードウェア化

ネットワーク層プロトコルはハードウェア化に適したプロトコルであるだろうか。ハードウェアをもちいてソフトウェア処理の実行することは、どのような処理でも可能である。しかし、ハードウェア化すればどのような処理でも高速化できわけではない。例えば、全ての処理間にデータ依存関係が存在するような処理をハードウェア化しても、処理時間は短縮されるであろうが、このような場合には積極的にハードウェアを利用する意味がない。

しかし、ネットワーク層プロトコルはハードウェアによって効率良く高速化できるプロトコルであるといえる。本手法のようにネットワーク層プロトコルを処理することで、入力として与えられるデータ間にデータの依存関係がないという特徴を活かしてハードウェア化できる。さらに、入力データがフィールドにわかれており、またその出現位置が固定であるため、ハードウェアでの処理分割が容易にできる。

例えば MIPS のような汎用プロセッサの場合は

表 5.1: MIPS のオペランド

	6 ビット	5 ビット	5 ビット	5 ビット	5 ビット	6 ビット
I 命令	オペランド	レジスタ名	レジスタ名	レジスタ名	シフト量	機能
R 命令	オペランド	レジスタ名	レジスタ名	アドレス		

と 2 種類の命令形式がある。この方式の場合、オペランドにより命令形式を変えるわけだが、オペランドを解析するまで下位 16 ビットが何を意味するのか解らないのである。

このように、ネットワーク層プロトコルはハードウェアによる処理に向いているプロトコルであると言える。

5.2 ハードウェアに適したプロトコルとは

ルータに要求されることは、到着したパケットをいかに早く送出するかである。つまり、ハードウェア処理上でクリティカルパスとなっている処理を如何に高速化するかが鍵である。IPv6 プロトコルで考えると、経路検索がクリティカルパスであるといえる。この処理を高速化するためにできることで、一番単純なことは、他のフィールドよりも早く処理を始めることである。つまり、いちばん先にハードウェアに入力される必要がある。本研究で IPv6 のハードウェアを設計した場合には、すべてのフィールドが同時に入力できると仮定したため、各モジュールが同時にスタートしたが、実際にはバスの幅等の制限から、フィールドが後ろの方になるほど転送に時間がかかると思われる。

つまり、ネットワーク層プロトコルをハードウェア処理に適したものとするために、簡単におこなえることは、フィールドの順番を入れ換えることである。その処理に

第 6 章

まとめ

6.1 まとめ

ネットワーク層プロトコルは非常に並列性の高いプロトコルであり、ハードウェア処理に適しているプロトコルであるといえる。そのため、ルータのパケット転送速度向上にはルータのハードウェア化が重要であると考え、ネットワーク層プロトコルをハードウェア化するさいの方法論を提案した。ネットワーク層プロトコルを一つの巨大な命令であるととらえ、VLIW 的に処理をおこなうハードウェアを構成した。提案した手法はプロトコル内のフィールドを単位として処理をおこなっている。そのため、各パケットに対して依存関係がないプロトコルであれば、本研究で提案した手法は十分適応可能であると思われる。

6.2 今後の課題

今後の課題として次のことがあげられる

- 例外処理にたいする制御法
- 具体的な処理時間評価法の考察
- ハードウェア仕様後におこなわれる、論理回路設計、論理合成などに対する方法論の考察。

謝辞

本研究を進めるにあたり、篠田陽一先生には日頃より適切な御指導、御助言をたまわり深く感謝致します。また、有益な議論をしていただきました、篠田研究室の諸兄に深く感謝致します。

さらに、本研究を進めるにあたり必要な機材、場所等を提供してくださり、有用な議論をしていただきました株式会社日立製作所の榎本博道主任技師をはじめ、ネットワーク部の皆様方に深く感謝致します。

参考文献

[1] David A.Patterson,John L.Hennessy, 成田 光彰 訳, コンピュータの構成と設計 (上) 日経 BP 社

[2] David A.Patterson,John L.Hennessy, 成田 光彰 訳, コンピュータの構成と設計 (下) 日経 BP 社

[3] John L.Hennessy,David A.Patterson, 村上 和彰 訳, コンピュータアーキテクチャ, 日経 BP 社

[4] Mike Johnson, 村上 和彰 監訳, スーパースカラ・プロセッサ, 日経 BP 出版センター

[5] R.Lipsett,C.Schaefer and C.Ussery, 杉山 尚志 監訳, VHDL, マグロウヒル 1990

[6] J.Reynolds,J.Postel ASSIGNED NUMBERS RFC1700,1994

[7] Deering.S and Hinden.R, Internet Protocol Version6 (IPv6) Specification, RFC1883,1995

[8] Deering.S and Hinden.R, Internet Protocol Version6 (IPv6) Specification, RFC2460,1998

[9] R, Hinden,S.Deering, IP Version 6 Addressing Architechture, RFC2373,1998

[10] Atkinson.R, IP Encapsulation Security Payload, RFC 1827,1995

[11] Atkinson.R, IP Authentication Header, RFC1827,1995

[12] 長谷川 裕恭 著, VHDL によるハードウェア設計入門, CQ 出版社 1995

[13] 斎藤 忠夫, 発田 弘 著, 高性能コンピュータアーキテクチャ, 丸善

付 録A

IPv6ヘッダ詳細

A.1　拡張ヘッダ

IPv6 で現在定義されている拡張ヘッダを以下に述べる。

- 中継点オプションヘッダ

  経路途中にある計算機で参照される。

- 終点オプションヘッダ

  終点アドレスとして指定された計算機において参照される。

- 経路制御ヘッダ

  通るべき中間ノードのアドレスのリストを含む。中間ノードで参照され、書き換えられる。

- 断片ヘッダ

  終点ですべての断片を再構成するために必要な情報を含む。また IPv6 では送信者以外の計算機は分割処理をおこなわない。

- 認証ヘッダ

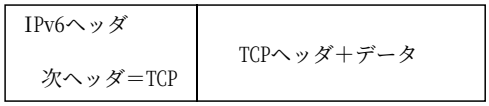
  データの正当性を明示的に保証する。

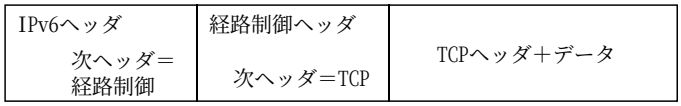
- 暗号ペイロードヘッダ

  機密性の確保をおこなう。

A.2　数珠つなぎヘッダ

拡張ヘッダは基本ヘッダから数珠つなぎ (daisy chain) 構造で挿入される。以下の二つはIPv6 基本ヘッダのみであった場合と、IPv6 基本ヘッダとIPv6 の拡張ヘッダである経路制御ヘッダが挿入された場合を示している。





A.3　アドレスの初期割り当て

IPv6 アドレスの初期値として割り当てられる値を次に表す。この値はIPv6 アドレスの先頭 1〜8 ビット目までの値である。

割り当て	プレフィックス
予約	0000 0000
未割り当て	0000 0001
NSAP 割当のために予約	0000 001
IPX 割当のために予約	0000 010
未割り当て	0000 011
未割り当て	0000 1
未割り当て	0001
ユニキャストアドレス	001
未割り当て	010
未割り当て	011
未割り当て	100
未割り当て	101
未割り当て	110
未割り当て	1110
未割り当て	1111 0
未割り当て	1111 10
未割り当て	1111 110
未割り当て	1111 1110 0
リンクローカル ユニキャストアドレス	1111 1110 10
サイトローカル ユニキャストアドレス	1111 1110 11
マルチキャストアドレス	1111 1111

表 A.1: アドレスの初期割り当て